(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 November 2004 (18.11.2004)

PCT

(10) International Publication Number
WO 2004/100496 A2

(51) International Patent Classification[7]: H04L 29/06

(21) International Application Number:
PCT/IB2004/051670

(22) International Filing Date:
2 September 2004 (02.09.2004)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: PISARAMEDIA OY [FI/FI]; Pentti Vataja, Liimassuontie 15, FIN-003300 Otalampi (FI).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: VATAJA, Pentti [FI/FI]; Kerokuja 3 A 6, FIN-01280 Vantaa (FI).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii)) for the following designations AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW, ARIPO patent (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG)*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii)) for all designations*

Published:

— *upon request of the applicant, before the expiration of the time limit referred to in Article 21(2)(a)*

— *without international search report and to be republished upon receipt of that report*

— *without classification; title and abstract not checked by the International Searching Authority*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: ENDS - MESSAGING PROTOCOL THAT RECOVERS AND HAS BACKWARD SECURITY

(57) Abstract: The presented messaging protocol uses three new public keys in a signed and encrypted message to achieve backward security and recovery in an environment where an attacker now and then obtains the security parameters in exposed, decrypted form. Backward security is understood to mean that an adversary cannot decrypt those captured encrypted messages that the user has decrypted prior the exposure. The recovery of the protocol means that the attacker at some point of time after the exposure cannot any more decrypt messages created after the exposure.The invention can be used e.g. in encrypted email communication. New to the current state of the art is that a message contains history data: a list of recently used public keys and their Diffie-Hellman counterparts. Also new is the usage of a stored and pseudorandomly changing data used together with a just computed Diffie-Hellman shared secret to provide a value that an attacker cannot produce if he does not have a proper exposed security data and the private key required to compute the Diffie-Hellman shared secret.

WO 2004/100496 A2

Description

ENDS - MESSAGING PROTOCOL THAT RECOVERS AND HAS BACKWARD SECURITY

Technical Field

[1] The invention is about encrypted communication and its protocol. A specific emphasis is placed on the backward security and recovery properties of the protocol. It is assumed that an adversary can now and then expose all of the cryptographical security data i.e. private keys and other data stored on participants' computers. The presented method tries to make the damages of security data exposure as small as possible. Practical example where the protocol can be used is encrypted email.

Background Art

[2] We address the problem of encrypted communication over insecure networks using computers whose contents can occasionally be studied by an adversary. Insecurity of the network means that it must be assumed that not necessarily every encrypted message reaches the intended receiver. The messages may be received at different order than they were sent. It is also expected that an adversary can now and then expose all of the cryptographical security data i.e. private keys and other data stored on participants' computers. The adversary then uses this exposed snapshot of security data (called shortly snapshot) to obtain as many previous or future plaintexts as possible. The presented method tries to make the damages of the security data exposure as small as possible.

[3] Practical example is encrypted email communication. Even if every message would reach its destination the user may process only some of them and those in an ad hoc order. The user should be able to enjoy the backward security concept: An exposure of current security data should not compromise previously decrypted data. Clearly also the recovery i.e. the regaining of the state where the adversary cannot decrypt messages created after the snapshot would be a great advantage. It must be assumed that the user has no knowledge of the attackers success.

[4] The usage of backward security was mentioned by Ross Anderson in an invited lecture in Fourth Annual Conference on Computer and Communications Security. ACM, 1997. Summary appears in: Two remarks on public key cryptology http://www.cl.cam.ac.uk/users/rja14/, 2001. Also the idea of protecting future communication after the exposure is discussed in the paper. According to Anderson in traditional (symmetric) systems, the backward security is provided by key updating: "two or more principals who share a key pass it through a one-way hash function at agreed times: $K_i=h(K_{i-1})$." Because of the one-way function the previous key cannot be derived from the current key.

[5] The future keys however can de derived by the attacker. To protect them "two or more principals who share a key hash it at agreed times with the messages they have

exchanged since the last key change: $K_{i+1} = h(K_i, M_{i1}, M_{i2}, \ldots)$". Backward security is still provided and also protection of future messages if the attacker misses one previous message. As noted by Anderson the advent of public key cryptography and Diffie-Hellman (DH) key exchange offers a stronger form of protection: when fresh public DH keys have been exchanged the security has been regained even if all the previous traffic is decrypted by the opponent. In this invention the idea that some kind of representation of previous keys affects new ones is used together with the possibilities that the DH key exchange offers.

[6] This recovery or freshness feature of the DH key exchange between two parties is widely used in the handshake phase of interactive session protocols. During the initial handshake the DH key exchange is performed and the shared secret is used as a security parameter for later message exchange. The individual messages during the session are then encrypted/decrypted on the basis of these parameters without any further DH exchange. If an attacker obtains the private key of one of the DH parties the whole session is exposed. This invention can also be seen as an approach to improve the security after the initial handshake. Please note that the communication need not necessarily be an online one. The presented protocol can naturally be used even if there are days between message movements - an example being email communication. Essential is that in the beginning the two communication parties are introduced to themselves via the protocol initialization.

[7] The current literature that deals with the problem of backward security in digital signatures and in encryption uses public keys in their more traditional meaning: one public key is distributed to many persons, which can then use it to send encrypted messages to the creator of the public key. The backward security methods in current literature are developed for this more general case. In this invention however the public keys used are intended to be used only between two specific and fixed communication parties - the fixed sender and the fixed receiver. In this sense our keys could be called session keys between two parties. In our method when two persons send a message to the same receiver both senders use different public keys of the receiver. The senders have no knowledge of the public key the other sender uses. The more general case where only one public key is reserved for many senders leads to more complex solutions. Please note that the current literature also uses the term forward-security in the same sense we use the term backward security: a current exposure does not expose old decrypted communication. The first method for the general case is R. Canetti, S. Halevi, J. Katz: A Forward-Secure Public-Key Encryption Scheme. EUROCRYPT 2003, LNCS 2656, 255-271. Springer-Verlag, 2003. Their method does not provide recovery. The recovery is added by Y. Dodis, M. Franklin, J. Katz, A. Miyaji, M. Yung: Intrusion-Resilient Public-Key Encryption. Topics in Cryptology - CT-RSA 03, Lecture Notes in Computer Science Vol. 2612, M. Joye ed, Springer-Verlag, 2003. This approach uses special update and refresh

messages.

[8] If the presented protocol is used in e.g. email communication each new contact must be processed through the protocol initialization phase. If the methods based on the more general usage of public keys are used this need not be done - however a public key must have somehow been delivered to the sender of the first message. Our initialization requires that both parties send one initialization message. Our protocol provides backward security and recovers when messages are exchanged.

[9] In symmetric encryption setting the problem of protecting old traffic is studied by M. Bellare, B. Yee: Forward-Security in Private-Key Cryptography. Extended abstract in Topics in Cryptology - CT-RSA 03, Lecture Notes in Computer Science Vol. 2612, M. Joye ed, Springer-Verlag, 2003. They recommend the usage of forward-secure pseudorandom bit generators to be used as a central primitive. This is our approach too. Their construction is such that the generator's input is a previous state (seed) and it generates a new state and the required random output bits. The old state is destroyed. If from the generated output bits it is infeasible to derive the previous state (seed) used, then the construction protects old outputs. To this construction we add another input: a Diffie-Hellman shared secret. This is added to provide the recovery feature. Our state is maintained in both computers and these states are required to go through same values during the communication.

[10] By pseudorandomness it is understood to mean that it is infeasible to derive the initial seed from the outputted bits or another outputted bit from another ones and that if the seed is unknown the outputted bits are unpredictable. One construction that can be used is based upon the Goldreich-Levin hard-core bit and Blum-Micali iteration, see O. Goldreich, L. Levin: A Hard Core Predicate for any One Way Function. Proceedings, 21st ACM STOC, 1989, 25-32. and M. Blum, S. Micali: How to Generate Cryptographically Strong Sequences of Pseudo-random Bits. SIAM Journal of Computing, 13. no 4, 1986, 850-864. Another PRG providing similar properties i.e. pseudorandom bits can be used instead of this generator.

[11] Let's now look the usage of DH exchange more closely. In traditional DH key exchange both parties send one message before the shared secret is computed and the actual encryption of the plaintext can be done. To provide optimal backward security and recovery one could do the DH exchange always before a plaintext should be encrypted. However, now only one third of the messages would carry information. Assume now that the attacker obtains one of the private keys. Then he could decrypt one cryptotext based on the obtained private key.

[12] If the messaging would always happen in turns - every message is replied - the DH exchange could be done so that every message carries one new public key for immediate DH calculation to decrypt the encrypted text in the message and one another new public key for the next message to be received. Every private key would be destroyed after the DH calculation. Every message would carry information and

again the attacker could only read one message based on one obtained private key.

[13] Problems will arise if a participant is allowed to send many messages before receiving a new public key. The sender of many consecutive messages must use the most recently received public key for all the messages he sends. If the attacker obtains the corresponding private key the backward recovery will be lost, the attacker can read all messages decryptable by this private key even if he obtains the key just before the last such message is decrypted.

[14] One development further would be to include a fixed number of new public keys in every message; the sender would prefer to use a public key only once. Suppose now that a sender will never use more messages than this fixed number before receiving a new set of public keys. Consider now the properties of such a system. There would be backward security and protocol would recover when the sender has received a new set of public keys. The presented invention will have backward security and recover at same time like this obvious fixed case but it uses only 3 new public keys in every message without a limit on the number of consecutive messages sent. One of the 3 keys is used for immediate DH calculation and two other ones are delivered to the message's receiver to be used in his next sendings.

Summary of Invention

[15] The presented protocol uses public key encryption and utilizes 3 new public keys in every message. These keys are randomly (pseudorandomly) generated i.e. randomness is collected from some source and pseudorandom generator produces the private keys, then the seed is destroyed. One of the public keys is used for immediate DH computation - two keys are reserved for future DH computations. New to the current state of the art is that a message may contain history information: a list of recently used DH public keys and identifiers of their DH counterparts. Also stored old data called state is used together with a pseudorandom bit genrator (PRG) when generating a new state and symmetric key - a symmetric key is produced not only by a result of DH shared secret computation, but also with the help of this old data. A block cipher uses the produced symmetric key to encrypt or decrypt one message.

[16] The used PRG must have following properties: the bits produced must be pseudorandom and unpredictable i.e. from the result it must be infeasible to derive the seed or another outputted bit from another ones, also if the seed is unknown it must be infeasible to derive the outputted bits. We arrange things so that the initial seed x_0 will be unknown to the attacker if he either misses a state or a DH shared secret - the seed x_0 is set to be the xor'd value of both of them.

[17] One PRG that can be used is described next. Now let f be a one-way permutation and r a random binary vector, x_0 is the initial seed, b(r,x) is a function that computes the Goldreich-Levin hard-core bit from x. Due to the Blum-Micali theorem of iterating a one-way permutation and Goldreich-Levin hard-core bit the n bits $b(r,x_0)$, $b(r,f(x_0))$, $b(r,f(f(x_0)))$ ...$b(r,f^{n-1}(x_0))$ are pseudorandom and unpredictable if x_0 is unknown, the

random vector r need not be unknown. The random vector r is chosen to be the DH shared secret. This PRG has the required properties. In this construction the required one-way permutation can be emulated by a block cipher's encryption operation, see the detailed description section.

[18] Now if the adversary wants to proceed forward but cannot produce x_0 he cannot produce the outputted bits. If the adversary wants to go backwards and has obtained the outputted bits and the DH shared secret he still cannot determine x_0. Neither can other outputted bits be derived from each other if some of them are revealed to the adversary.

[19] The history information in messages is needed because it enables the states being produced in forward proceeding order even if messages are not decrypted in order. In such a case the history data: a list of recently used DH public keys and identifiers of their DH counterparts - is read and used to produce the states and keys in order. A produced key is stored together with its message number. When a message corresponding to this number should be decrypted the key is fetched from the store and then removed. Only the minimum amount of history information needed is included in a message. When a message is decrypted the history data to be placed in the next outgoing message can be decided. The history data although consisting of public keys is encrypted - this is done to prevent a casual observer from doing any conclusions from it. This encryption key is derived from the DH shared secret without the stored state.

[20] When encrypting we do not store the private key used to produce the DH shared secret. This implies that the sender cannot decrypt his own sent messages but neither the adversary can decrypt outgoing messages based on a snapshot from sender's computer.

[21] The protocol has recovered when the last snapshot the attacker has does not contain the private key whose corresponding public key will be used in the next encrypted message to be received. Consider the case where the persons involved encrypt and decrypt messages in turns (they decrypt a message and then send a reply) and the snapshot is obtained from one computer. The adversary can decrypt depending on the timing of the snapshot either 0 or 1 message until the protocol recovers. If messages are not exchanged in turns but many messages are received before a new one is sent it may also happen that adversary can't decrypt a single message. If the timing of the snapshot is better messages can be decrypted until the sender receives new public keys.

[22] Two of the new public keys in a message are intended for the receiver of the message to be used in his next sendings as DH counterparts. One of the keys is used only once in a DH calculation, the other one is used many times if no newer public keys are available. The usage of the one-time key prevents in certain situations the attacker of using an older snapshot in order to decrypt more messages than a newer snapshot enables.

[23] All the messages are signed using the digital signature method of the underlying public key scheme. The private/public key pair used is selected to be the other one than the one-time one of the latest public keys the intended receiver is known to posses. The plaintext ends with a mac-value. The PRG produces the corresponding mac-key together with the encyption/decryption key. The verification of the mac ensures that proper symmetric keys were used and the plaintext was created by the sender and was decrypted successfully. Note that the protocol relies on the states on different computers being updated in synchronous way, thus e.g. a restored backup of the security data may destroy this dependency.

[24] As extra countermeasure against certain kinds of attacks the two new public keys in a message can be encrypted together with the plaintext. Note that then an attacker cannot know a DH counterpart unless he has decrypted a previous message or obtained a proper snapshot. This provides protection if the underlying public key scheme or its implementation has weaknesses. Note also that the protocol itself without this encryption of public keys provides same kind of protection: a DH shared secret alone is not sufficient to produce a symmetric key - the previous stored state is also required.

[25] The underlying public key scheme might be a RSA one or based on elliptic curves or some other scheme. On elliptic curves the public key generation can be done on tens of milliseconds even if currently adequate field size is used.

[26] To decrypt more messages the attacker may want to launch a so-called man in the middle attack. This is easiest to the attacker if he succeeds in obtaining the snapshots of both communication parties' security data at the same time. Lets assume the parties involved are Alice and Bob, the adversary being Charlie. Alice now sends a message to Bob. Charlie captures the message and uses the snapshot of Bob's data to decrypt it and then uses Alice's snapshot to encrypt it again for Bob and only then passes the message to Bob. This can continue as long as the adversary wishes. Every message from Bob and from Alice can and must be decrypted/encrypted. Another way to start this kind of attack is to modify the security data on Alice's computer and send a message to Alice pretending it being from Bob or to create a message to Bob based on the knowledge obtained from a snapshot from Alice's computer. The man in the middle attack can be detected if a cryptographic hash value (used as a checksum) is computed from the message. The hash value on Alice's computer of a message sent from Alice's computer will differ from that of at Bob's computer if there is an active man in the middle attack. If the adversary wants to stop this attack and not arouse any suspicion he has to modify the security data on at least one of the computers involved. If during active man in the middle attack one of the messages is not captured and changed by the attacker the receiver will not be able to verify its signature.

Detailed Description

[27] A message has three parts: public, history and the text part.

[28] The public part consists of: identifier of the receiver, message number, a new public

key (called refresher), identifier of the refresher's Diffie-Hellman counterpart (called DH identifier), 2 new public keys (called one-time and repeater) and the message text part's starting position. As an option the one-time and repeater public keys can be placed together with the message's text and be thus encrypted.

[29] History part: number x, a list of x number of items each consisting of a public key and the corresponding DH identifier (this list of items is called history data).

[30] Text part: the message text, optionally the one-time and the repeater public keys may be placed here, a mac-value.

[31] The message ends with a digital signature of the whole message.

[32] New to the current state of the art is that a message contains history data: a list of recently used public keys and their DH counterparts. Using this history data it is possible to have a state in both the sender's and the receiver's computer that goes through same values even if some of the messages sent are never received or not decrypted. The reader should also notify the usage of two different kinds of public keys: a one-time one and one that is used if no newer public keys are available.

[33] The public part is not encrypted, others are but with different symmetric keys. The history part's key is derived from the public part's DH shared secret without the use of the so-called key generator. The text part's key is derived using the key generator. A block cipher performs the encryptions/decryptions.

[34] A security data storage (store) hosts for each messaging contact:

[35] Sent keys collection (a public and private key, a message number that carried the keys)

[36] Signature keys collection (a public key for signature verification and a message number)

[37] Waiting keys collection (a message number, a public key for signature verification, a symmetric key and a mac-key)

[38] History data items (each item consisting of a message number and a sent public key and the DH identifier of the corresponding public key)

[39] Two states: one for sending and one for receiving (a state has two blocks b_0 and b_1, their size is the size of a DH shared secret).

[40] Also stored are:

[41] Latest received one-time and repeater public keys and the message's number that carried them

[42] Latest sent repeater key the receiver is known to posses.

[43] This storage is in user's computer typically in encrypted form and a password or a passphrase is needed for its use. When the protocol is used for encryption or decryption this storage must be available to the protocol. If an attacker succeeds in obtaining the data in this storage in decrypted form he is said to have obtained an exposed snapshot (shortly snapshot) of the security data.

[44] A key generator produces a new state, the symmetric key for encryption or

decryption and the mac-key for plaintext verification. The encryption/decryption key and mac-key are called shortly symmetric keys. Part of the key generator's input comes from the security data storage: the state, the other part of the input is a newly computed DH shared secret. The key generator uses a pseudorandom bit generator (PRG) to produce its output. We arrange so that the initial seed x_0 to the PRG will be unknown to the attacker if he either misses a state or a DH shared secret – the seed x_0 is set to be the xor'd value of both of them. This is a new construction compared to the current state of the art.

[45] The used PRG must have the following properties: the bits produced must be pseudorandom and unpredictable i.e. from the result it must be infeasible to derive the seed or another outputted bit from another ones, also if the seed is unknown the outputted bits must be infeasible to derive.

[46] We next describe one possible PRG.

[47] Let GL(r,x) be a function computing the Goldreich-Levin hard-core bit from bit vectors x and r, r and x being of equal length. The GL bit is defined to be the inner product of x and r modulo 2: $(x_1r_1+x_2r_2+...+x_nr_n) \bmod 2$, where the indices are the corresponding bits of x and r.

[48] Let Enc(key,plaintext) be an encryption operation of a block cipher.

[49] Let blocks b_0 and b_1 be the old state, k be the number of bits in the old state and in the symmetric keys. Let dr_i be the i'th derived pseudorandom bit and x a bit vector.

[50] Key generator - derive the new state and the symmetric keys:

[51] Produce a block unknown to the attacker if he misses b_0 or DH shared secret:

[52] $x = b_0$ xor DH shared secret

[53] Run the PRG:

[54] For i=1 to k

[55] dr_i = GL(DH shared secret,x)

[56] $x = Enc(x,b_1)$

[57] End for

[58] The produced bits dr_i ,i=1...k will form the new state and the symmetric keys. Note that the usage of the Enc-function is the usual way to construct a one-way hash function using a block cipher: to compute a one-way hash of x compute Enc(x,p) where p is some fixed known plaintext. In our case the fixed plaintext varies between invocations of this PRG but stays the same during a specific invocation. If the attacker obtains a state s_m and the next state s_{m+1} but misses a DH shared secret he still cannot run the PRG forward from s_m. From the produced state s_{m+1} due to the pseudorandomness of the hard-core bits he cannot derive other produced bits (the symmetric keys) or the initial seed x. If the b_1 would be the same during all invocations of this PRG (if b_1 is hard coded in the program) the attacker could without obtaining a snapshot perhaps try to guess the initial x (the result of the xor-operation). Now he has to obtain the snapshot and guess the initial x or without a snapshot guess both initial x

and b_1. The properties of this PRG allow the usage of the block b_1 and the DH shared secret in producing the outputted bits without compromising the unpredictability and pseudorandomness of the results.

[59] Consider now sending a message. Three new public keys are generated (called one-time, repeater and refresher). These keys are randomly (pseudorandomly) generated i.e. randomness is collected from some source and a pseudorandom bit generator produces the keys, then the seed is destroyed. The used PRG can be of a Blum-Micali Goldreich-Levin type or a different one - essential is that the seed is destroyed and that a private key cannot be determined from another one.

[60] This message's number is set to be one greater than the latest one sent. The generated one-time and repeater public keys are placed into their places in the message and the one-time private key and the repeater private key are stored in the sent keys collection. The refresher key is placed into its place in the message, however its private key is not placed into the store.

[61] From the store the message receiver's latest one-time and repeater public keys are fetched together with the message's number that carried them. If the receiver's one-time key has not been used before when sending then it is used as the DH counterpart otherwise the receiver's repeater key is used. The DH shared secret is computed with the receiver's selected key and with the sender's refresher key. In the public part of the message the DH identifier is set to be the message number that carried the DH counterpart and a value indicating whether the one-time or repeater key is used. When the shared secret has been computed the private key of the sender's refresher key is destroyed without it being stored. This will have the implication that the sender cannot decrypt messages sent from him, but neither can the adversary use a snapshot to decrypt outgoing messages. Adversary's possibilities are limited to incoming messages.

[62] Next from the store the history data items for the receiver are fetched and copied into the message. A symmetric key is derived from the DH shared secret (by e.g. computing its hash value) and the history part of the message is encrypted with this key. The new history data to be stored is build by adding this message's number and the refresher public key and the identifier of its DH counterpart to the top of the old history data. The message text part's starting position field in the message's public part is adjusted based on the size of the history part.

[63] Next the key generator is used the input being: the state for sending messages and the just computed DH shared secret. The new generated state is stored and the outputted symmetric mac-key is used to calculate text's mac-value (if the option of placing the one-time and repeater public keys together with the text is used then they are also included in this mac) and a block cipher encrypts the message text part using the produced encryption key.

[64] The whole message is digitally signed. The private/public key pair used in signing

is selected to be the sender's latest repeater key the receiver is known to posses. The receiver's latest repeater public key and this message's number that is being sent are stored in the signature keys collection.

[65] Consider now decrypting a message. Based on a comparison between this message's number and the greatest number decrypted so far we consider three cases: 1) This message's number is one greater; 2) This message's number is two or more greater; 3) This message's number is less.

[66] If cases 1 or 2 apply then a message number x is extracted from the public part's DH identifier. From the store's signature keys collection a public key is fetched based on this number x. The signature at the end of the message is verified using this public key. If the signature does not verify the decryption is abandoned.

[67] The private key identified from the DH identifier is fetched from the store's sent keys collection. The DH shared secret between the fetched key and message's refresher public key is computed and a symmetric key is derived from it in order to be able to decrypt the history part of the message.

[68] If we are on case 1 then the state for receiving is fetched from store and the key generator uses the state and the DH shared secret to generate the new state and to output the symmetric keys. The outputted symmetric key is used by a block cipher to decrypt the message's text part. The mac-key produced is used to calculate a mac-value of the plaintext. The calculated mac-value is checked against the value found after the plaintext and if found being different the decryption phase is abandoned. The new state is stored. The received new public keys (one-time and repeater) are now the latest ones and they are stored. The latest repeater key the sender is known to posses is now set to be the repeater key that was in the message x when x was sent (the key is fetched from the sent keys collection).

[69] If we are on case 2 there are messages that have not been decrypted between the latest decrypted one and this message. The history part of this received message is decrypted and the key generator will be run to produce the symmetric keys for each message between the latest decrypted one and this message. From the history data list the first not yet processed refresher key and its DH identifier are identified based on the message number in the DH identifier. The list is processed each item in turn and the key generator is run to produce the next state and the symmetric keys. At the same time also a public key is determined that must be used to verify the signature of the message in question. For each message the generated keys and signature verification public key will be stored in the waiting keys collection together with the message number in question. This storing into the waiting keys collection will actually take place only after this message's mac-value has been verified. When the last item in history data has been processed the current message's DH shared secret and the state produced by the iteration on history data are inputted to the key generator. The outputted symmetric keys are used to decrypt and verify this message's text. If the

mac-value is not verified no storing is performed and the decryption is abandoned otherwise the storing into the waiting keys collection is performed and the last generated state is also stored. The received new public keys (one-time and repeater) are now the latest ones and they are stored.

[70] If cases 1 or 2 apply then it may happen that after the decryption several items in the store can be removed. Let x be the message number extracted from this message's public part's DH identifier. Sent keys collection: an item with message number less than x is removed. The one-time private key of message number x is removed if it exists in the collection. History data: an item whose message number is less or equal to x is removed. Signature keys collection: every item whose message number is less than x is removed.

[71] If case 3 applies the store's waiting keys collection is used to provide the signature verification public key and the symmetric key for the message text's decryption. After signature verification and the text part's decryption and plaintext's mac value's verification the information relating to this message number in the waiting keys collection is removed. Please note that neither the history part nor the refresher key is studied in this case. The message text's starting position can be determined from the corresponding field in the public part of the message.

[72] To clarify the effect of using the one-time key consider the following situation: Charlie obtains a snapshot of Alice's security data before Alice encrypts a message to Bob. The private keys of the new public keys in the Alice's message are thus not in the obtained snapshot. Now Bob responds with many messages to Alice before decrypting a newer message from Alice. Alice decrypts the replies in order the first one being the message x. Now Charlie obtains another snapshot before Alice decrypts the message number y in the middle of Bob's responses. Charlie has now the state and private key needed to decrypt Bob's message y and messages from y forward. However, Charlie cannot go backwards in the state chain and thus the message x and messages before y cannot be decrypted based on the newer snapshot. The older snapshot exists but the problem from Charlie's viewpoint is that when Alice decrypted the first message x she destroyed the one-time private key. Charlie cannot proceed forward from the older snapshot's state since a private key is missing. If there would not be this one-time key concept the private key used to decrypt messages x-y would enable Charlie to proceed forward from the older snapshot. If Alice would not decrypt the messages in order the situation would be still the same. The private key of the one-time key is destroyed when the state is iterated forward when processing the history data of a message.

[73] During the initialization of the protocol both parties send to each other a one-time public key and a repeater public key with message number 0. The keys are stored into the sent keys collection with message number 0. A receiver of a repeater public key stores it into the signature key collection with message number 0. The receiver of a one-time and repeater keys stores them as the latest ones. Store's latest sent repeater

key the receiver is known to posses is set be the just sent repeater key.

[74] The states' initial values are determined by a DH exchange, this public key is in the same message that delivers the first one-time and repeater keys. Note that the number of blocks in the state depends on the requirements of the used PRG. In the following we assume that the state consists of two blocks, if this is not the case the arrangements can be easily altered according to the number of blocks used. Let c_1, c_2, c_3 and c_4 be fixed blocks of data, their size being the size of the state's block and let Enc(key,plaintext) be an encryption operation of a block cipher. If the public key used to initialize the states is bigger than the other public keys then a one-way hash function is used to produce either one block dh_1 or two blocks dh_1 and dh_2 from its DH shared secret. If there is one block dh_1 available then Alice's state for sending to Bob is b_0 =Enc(dh_1,c_1), b_1=Enc(dh_1,c_2) which is also Bob's state for receiving from Alice. Bob's state for sending to Alice is b_0=Enc(dh_1,c_3), b_1=Enc(dh_1,c_4), which is also Alice's state for receiving from Bob. If two blocks dh_1 and dh_2 are available then Alice's state for sending to Bob is b_0=Enc(dh_1,c_1), b_1=Enc(dh_2,c_2) which is also Bob's state for receiving from Alice. Bob's state for sending to Alice is b_0=Enc(dh_1,c_3), b_1=Enc(dh_2,c_4), which is also Alice's state for receiving from Bob.

[75] To convince the parties that the initialization messages are not altered they either a) are accompanied with a certificate and signed with the public key the certificate certifies or b) the parties compute a cryptographic checksum (hash value) of the message sent and received and ensure each other that the checksum is the same at both ends – this can be done by e.g. using voice contact.

[76] Please note that the history part of a message varies in size when messages are sent and received. Although the history part is encrypted a casual observer might draw some conclusions based on its size. To prevent this the history part may be set to have a specific predetermined fixed size and if the actual required size exceeds this then a random value from some predetermined range is added to the required size. Let x be the most recently sent message by Alice. A message received by Alice which uses one of the public keys in message x as DH counterpart empties the history data of Alice. The next message Alice sends would have history data of zero size. The above-described method can be used to hide e.g. this information.

[77] Please note that the message number in the public part of the message reveals how many messages the sender has created. If a casual observer then sees messages he can draw some conclusions on the messaging behaviour of the parties. To hide the starting point of the message number it can be started from some number that is derived from the DH shared secret in the protocol initialization phase.

[78] Note also that if the one-time and the repeater public keys are placed in the message's text part then an adversary has no knowledge of a refresher key's DH public counterpart unless he has succeeded in decrypting the required message or has obtained a suitable snapshot. This provides protection if the underlying public key

scheme or its implementation has weaknesses. The protocol itself also gives same kind of protection since a solved DH shared secret is not sufficient to produce a symmetric key - the proper stored state is also required.

[79] The message's public part contains the starting point of the message's text part thus revealing the size of the history part. It is possible not to place this starting point information in the public part and thus avoid using the technique to hide the size of the history data. The solution is to add this starting point information into a list item of the history part's list. Now every list item contains also the starting point of the corresponding message's text part. During encryption operation when a new list item is added on top of the history data the current message's text part's starting point is stored into the added item.

[80] If during a decryption the current message's number is seen to be greater than the latest decrypted one then the size of the current message's history part can be determined by decrypting the first one of its blocks, now the stored number - which tells how many items there are in the list - is used to compute the size of the history part.

[81] If during a decryption the history part is used to produce symmetric keys then this starting point found in a list item is stored into the waiting keys collection together with the symmetric keys and the signature verification key. When the waiting keys collection is used to decrypt a message this stored starting point tells where to start the decryption of the text part. Using this solution there is no need to try to hide the history part's size from a casual observer.

Claims

[1] A communications protocol comprising the parts of:

a) a message consisting of three parts: public part, history part and text part; public part consisting of a message number and 3 public keys called refresher, one-time and repeater, the public part includes also the refresher key's Diffie-Hellman counterpart's identifier and the starting point of the text part; the history part consisting of a number and a list of items, an item consisting of a public key and an identifier of its Diffie-Hellman counterpart;the text part consisting of a text and a mac-value

b) a store in both participants computers, storing for each contact: one state for receiving and one for sending and history data and private and public keys and symmetric keys

c) a pseudorandom bit generator PRG(x_0,b_1 ... b_n,dh,k), its input consisting of the seed x_0, blocks b_1 ... b_n, a block dh and number k; its output being k bits; the outputted k bits having the property of being pseudorandom and unpredictable if the seed x_0 is unknown and the parameters b_1 ... b_n and dh are not unknown

d) a key generator, its input being a state consisting of blocks b_0 ... b_n, and a Diffie-Hellman shared secret dh; its output being k bits that are produced using the pseudorandom bit generator PRG according to:

$x=b_0$ xor dh

PRG(x,b_1 ... b_n,dh,k)

the PRG producing the outputted k bits that form a new state and a symmetric encryption key and a symmetric mac-key

e) encryption operation consisting of generating 3 new public keys called refresher, one-time and repeater and computing a Diffie-Hellman (DH) shared secret between the refresher key and message receiver's latest one-time or repeater key, the one-time key being the preferred one but used only once, and inputting the DH shared secret and a state for sending to the key generator of part (d) and producing a new state and symmetric keys for message text's mac-value computing and encryption using a block cipher, and copying a stored history data to the history part of the message and storing the history data adding on top to it a new item consisting of the public refresher key and the identifier of its DH counterpart, and deriving a symmetric key from the DH shared secret and encrypting the history part of the message with this key and digitally signing the message using a previously sent repeater key that is the latest one the receiver is known to posses, and storing the new state and private keys of the one-time and repeater key but not storing the private key of the refresher key

f) decryption operation consisting of determining:

f1) if the received message is the latest decrypted one plus 1 then verifying the signature of the message and calculating a DH shared secret between

the message's refresher key and its DH counterpart and inputting this and the state for receiving to the key generator of part (d) and producing a new state and symmetric keys for message text's decryption, mac-value calculation and verification,

f2) if the message is the latest decrypted one plus 2 or more then verifying the signature of the message and calculating a DH shared secret between the message's refresher key and its DH counterpart and deriving a symmetric key from the DH shared secret and decrypting the history part of the message with this key and using the history part of the message and the key generator of part (d) to produce the required symmetric keys for each message between the latest decrypted one and this current message and also determining the signature verification public key for each of these messages and storing these values if the received message's text part's mac-value is verified,

f3) if the received message has not been decrypted and its message number is less than the latest decrypted one then its stored signature verification public key and stored symmetric keys are used to verify the signature, decrypt and verify the mac of message's text, after successful decryption the used information being deleted from store

f4) if the received message is the latest decrypted one plus 1 or more then after a successful and verified decryption the stored history data, private and public keys are purged based on the DH counterpart of the refresher public key in the message using the rule: if x is the message number in the DH counterpart then from the history data every item that was in a message numbered less or equal to x is removed and private keys whose corresponding public keys were in sent messages numbered less than x are removed and the one-time private key that was in the message x is removed if it still exists in the store

g) initializing the communication between the participants by both of them sending to each other a one-time and repeater public key and a third public key for generating a DH shared secret from which the initial states for sending and receiving are derived, the sent messages either being signed by a certified public key of the sender or the cryptographic hash value of the sent messages being verified of being equal by e.g. using voice contact.

[2] An apparatus as defined in claim 1, where the pseudorandom bit generator $PRG(x_0,b_1,dh,k)$ produces the outputted bits according to:

let Enc(key,plaintext) be an encryption operation of a block cipher and GL(r,x) be a function to extract the Goldreich-Levin hard-core bit from block x using a random block r, dr_i being one produced bit, the k outputted bits produced according to:

$x = x_0$
For i=1 to k
$dr_i = GL(dh,x)$
$x = Enc(x,b_1)$
End For.

[3] An apparatus as defined in claims 1 or 2, where
i) the size of the history part in a message is either exactly the size of the requir ed size or the size of the history part is of a predetermined fixed size and if the actual required size exceeds this then a random value from some predetermined range is added to the required size to make the final size
ii) the message number in the public part of the message is either started from the number one or is started from a number that is derived from the DH shared secret in the protocol initialization part (g)
iii) the one-time and repeater public keys are either in the message's public part or are in the message's text part.

[4] An apparatus as defined in claim 1, where
i) the message's public part does not contain the starting point of the text part and the message's history part's list item consists of a public key and an identifier of its Diffie-Hellman counterpart and a starting point of a text part
ii) during encryption operation (e) when a new item is added on top of the history data then the item consists of a public refresher key and the identifier of its DH counterpart and the starting point of the current message's text part
iii) during decryption operation (f) if in phase f1 the size of the current message's history part is determined by decrypting the first block of the history part and then using the number in the history part to compute the size of the history part
iv) during decryption operation (f) if in phase f2 the size of the current message's history part is determined by decrypting the first block of the history part and then using the number in the history part to compute the size of the history part and then when the history part is used to produce the symmetric keys for messages between the latest decrypted one and the current one then the starting point of the text part obtained from a list item is stored together with the symmetric keys and the signature verification public key
v) during decryption operation (f) if in phase f3 then the stored starting point of the text part is used to determine the current message's text part's starting point.

[5] An apparatus as defined in claim 4, where the pseudorandom bit generator $PRG(x_0,b_1,dh,k)$ produces the outputted bits according to:

let Enc(key,plaintext) be an encryption operation of a block cipher and GL(r,x) be a function to extract the Goldreich-Levin hard-core bit from block x using a random block r, dr_i being one produced bit, the k outputted

bits produced according to:

$x = x_0$

For i=1 to k

$dr_i = GL(dh,x)$

$x = Enc(x,b_1)$

End For.

[6] An apparatus as defined in claims 4 or 5, where

i) the message number in the public part of the message is either started from the number one or is started from a number that is derived from the DH shared secret in the protocol initialization part (g)

ii) the one-time and repeater public keys are either in the message's public part or are in the message's text part.